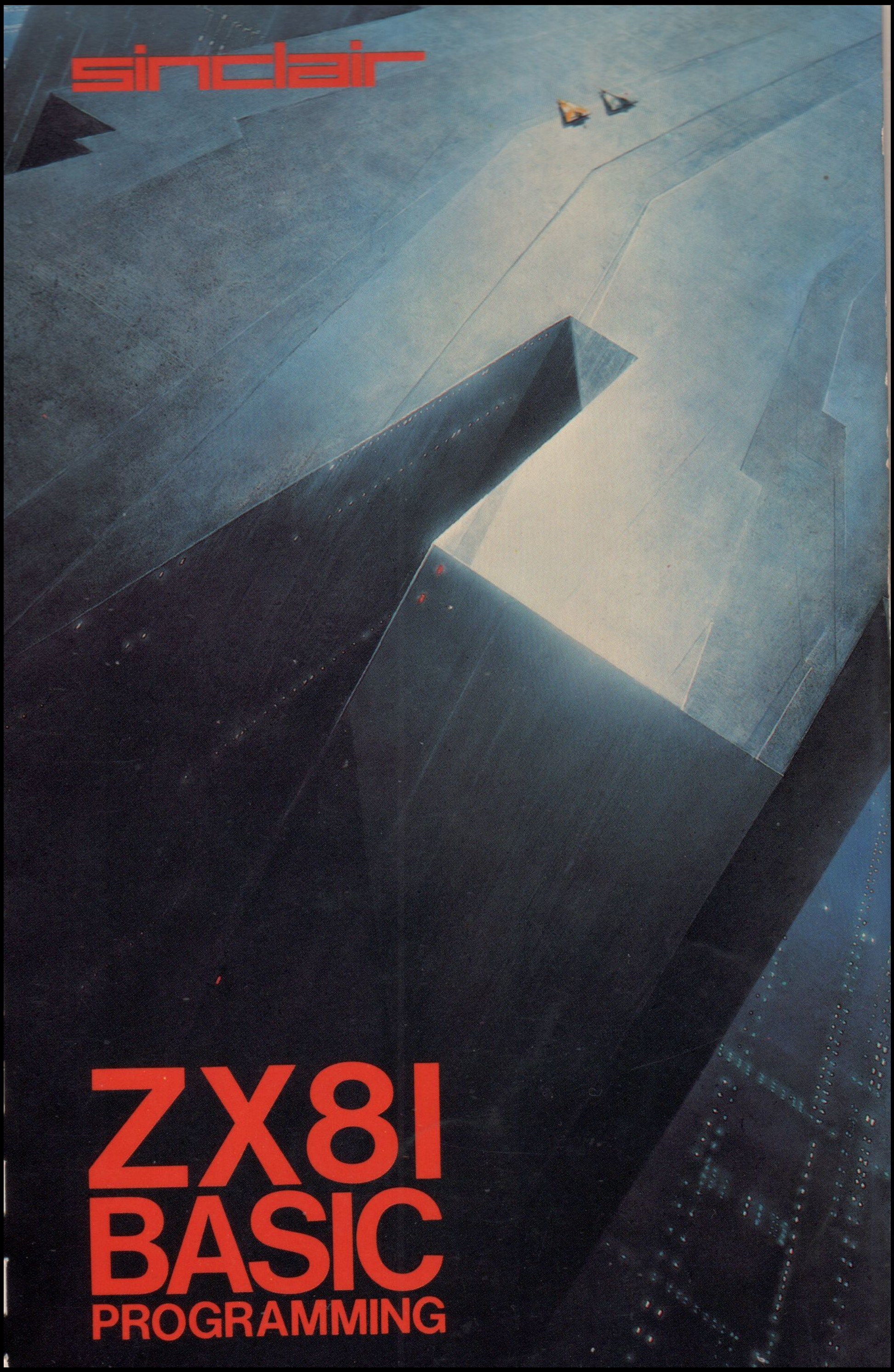
ZXZINE

Issue #8 November 2021



ZXzine

Published by:
Timothy Swenson
swenson_t@sbcglobal.net
swensont@lanset.com

ZXzine is published as a service to the Sinclair ZX81 community. Writers are invited to submit articles for publication. Readers are invited to submit article ideas.

Created using Open Source Tools:

- OpenOffice
- Scribus
- Gimp
- SZ81
- Zesarux

Table of Contents

Editorial 1

Active T/S 2068 Group 1

Expanding T/S 2068 Software 1

Porting Assembly from Spectrum to T/S 2068 1

tsmake 2

Boriel's ZX BASIC Compiler & T/S 2068 3

zxpaint2 3

More Astronomical Programs for the ZX81 4

Prime Number Benchmark 5

Structured Programming with ZX81 BASIC 6

Editorial

It's been a couple of years since the last issue of ZXzine. Working from home just extended by working day and limiting my time for hobbies. I've been working on these articles for a while, but here they are.

There are articles for both the ZX81 and T/S 2068. I might start working on more T/S 2068 programming projects, now that there are more tools for T/S 2068 programming. With the T/S 2068 having more colors and higher resolution than the ZX81, it takes more work to make a T/S 2068 program look nice than it does for the ZX81, which means more work on making things look pretty. Since programmers are lazy, I hope that I don't let that affect working on T/S 2068 programs.

Active T/S 2068 Group

David Anderson has put together www.timexsinclair.com. It is a web site for all Timex/Sinclair products, user groups, newsletters, etc. The focus is on the T/S products, but the ZX81, QL and Z88 are mentioned.

Through him I found out about an active T/S 2068 mailing list hosted on groups,io. The e-mail address to join the group is:

TS2068+owner@groups.io

Expanding T/S 2068 Software

With the active T/S 2068 group, there has been some discussion of getting more software to the T/S 2068. A lot of discussion was on porting Spectrum games to the T/S 2068, but that takes a fair amount of work, one game at a time.

Since the T/S 2068 and Spectrum are mostly compatible at the BASIC level, I thought it would be easy to find Spectrum programs that were probably written in BASIC and test them on the T/S 2068. I was already looking at Othello/Reversi programs for the Spectrum, so I tried out the different programs on the T/S 2068. I also had some Spectrum Astronomy software. I will create a list of the software that I have tried and post it on my website. As I run across more software to test, I will update the list. The World of Spectrum archive has quite a number of archived games to try out. There are two versions of World of Spectrum:

https://worldofspectrum.org/
https://worldofspectrum.net/ (World of Spectrum Classic)

Another source of programs are Type-In's from different magazines. The Type-Fantastic web site (http://www.users.globalnet.co.uk/~jg27paw4/type-ins/typehome.htm)
is a repository for listing type-in programs for the Spectrum, ZX81 and QL, from a number of UK magazines. The list has information about the different programs and if they are in BASIC, Machine Code, or a combination of both. There are a number of programs that are in just BASIC. I've tried some of these and most have worked with no issue. There is a whole trove of programs to look at, with most being a game of some sort. It is likely that the magazines that the programs came from are available via the Internet Archive.

I thought it would be a good idea to do something similar for T/S 2068 programs, so I have gone through Time Designs and UPDATE! magazines and created a list of type-in programs. I will post this on my website and update it if I find other newsletters to add. I have typed in a few of the programs and they will also be on my website.

Porting Assembly from Spectrum to T/S 2068

In the last year I have discovered two versions of the T/S 2068 ROM disassemblies. The Spectrum ROM has long been available in a couple of forms. Using these documents, I have been able to port some Spectrum assembly language programs to the T/S 2068.

Since the ROM's are a little different between the systems, the porting process is to convert all Spectrum ROM calls to T/S 2068 calls. One of the T/S 2068 ROM documents has a list of Spectrum calls and the equivalent for the T/S 2068. The problem is that some of the Spectrum calls are not the well published ones. This means that I have to look at the code in both ROM's looking for exact matches.

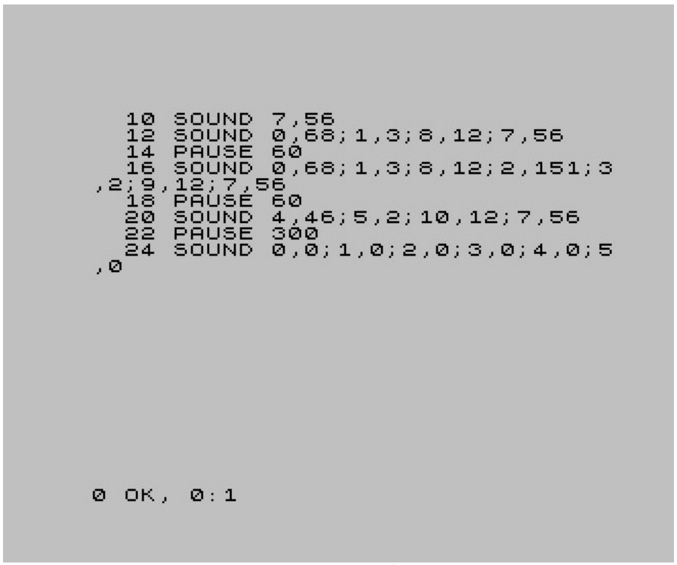
I have documented this process in a paper and published it on my website.

tsmake

zmakebas is a tool for converting BASIC text files into a .tap file for the Spectrum. Since the Spectrum BASIC is 99% the same with the T/S 2068 BASIC, zmakebas works for the T/S 2068.

There are some T/S 2068 keywords that zmakebas does not support. The most important of these is the SOUND command. Others are ONERR, STICK, FREE, and RESET.

Looking at the Spectrum and T/S 2068 character sets, SOUND is the same as the right brace (}), ON ERR is the same as the left brace ({), RESET as the asterisk (*), STICK the same as the pipe symbol (|) and FREE the same as the tilde symbol (~).

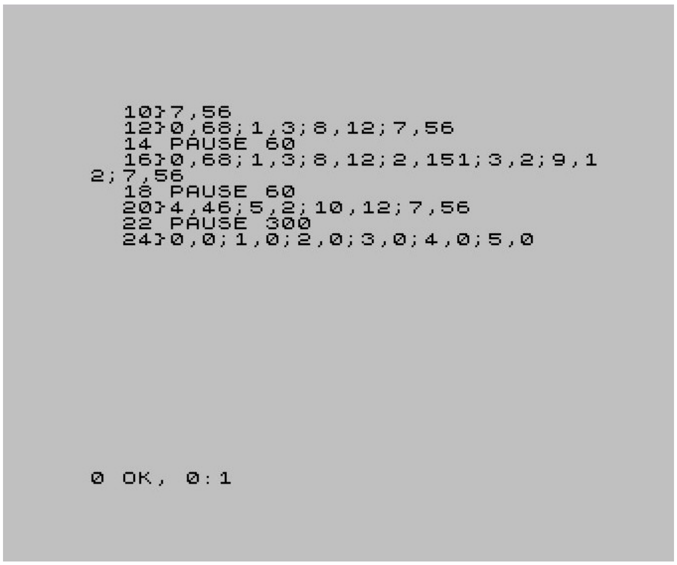


```
   10 SOUND 7,56
   12 SOUND 0,68;1,3;8,12;7,56
   14 PAUSE 60
   16 SOUND 0,68;1,3;8,12;2,151;3
,2;9,12;7,56
   18 PAUSE 60
   20 SOUND 4,46;5,2;10,12;7,56
   22 PAUSE 300
   24 SOUND 0,0;1,0;2,0;3,0;4,0;5
,0




0 OK, 0:1
```

Program on T/S 2068

I found that if I put in a } symbol, ran zmakebas, then loaded the TAP file into an emulator in T/S 2068 mode, I would get the SOUND keyword. The same TAP file into an emulator in Spectrum mode and I got the right brace.

Now, I could look into the source code of zmakebas and see how to have it find the SOUND command and put in the right character in the TAP file, but it could take a bit of work to get that done. All I really needed to do was to take the SOUND work and convert it to the right brace. I decided to write a shell script that would do that and then run zmakebas. All the user would have to do is to run the shell script with the basic text file as an argument and they would get a TAP file out the other end.



```
   10}7,56
   12}0,68;1,3;8,12;7,56
   14 PAUSE 60
   16}0,68;1,3;8,12;2,151;3,2;9,1
2;7,56
   18 PAUSE 60
   20}4,46;5,2;10,12;7,56
   22 PAUSE 300
   24}0,0;1,0;2,0;3,0;4,0;5,0




0 OK, 0:1
```

Program on ZX Spectum

The shell script is called tsmake.sh. It converts the 5 "new" keywords in a .bas file, to the right characters for zmakebas to process into a TAP file. The script creates a temp file for to hold the basic file after it has been modified. I did not want to modify the original basic file as that would affect the readability of the original program. The script then feeds the temp file into zmakebas. The arguments for zmakebas are hard-coded into the script, but the user can easily edit them.

```
#!/bin/bash
# tsmake.sh
# Converts T/S 2068 keywords to
character that zmakebas
# can handle and then runs
zmakebas.
# Keywords are: SOUND, ON ERR,
RESET, STICK, FREE
#   NOTE: If these words are used
in a strings, they will also be
changed.
```

```
#
# usage:  ./tsmake.sh test

cp $1.bas $1.basx
sed -i 's/sound/}/gI' $1.basx
sed -i 's/on err/{/gI' $1.basx
sed -i 's/onerr/{/gI' $1.basx
sed -i 's/reset/\\*/gI' $1.basx
sed -i 's/stick/|/gI' $1.basx
sed -i 's/free/~/gI' $1.basx
zmakebas -o $1.tap $1.basx
rm $1.basx
```

Boriel's ZX BASIC Compiler & T/S 2068

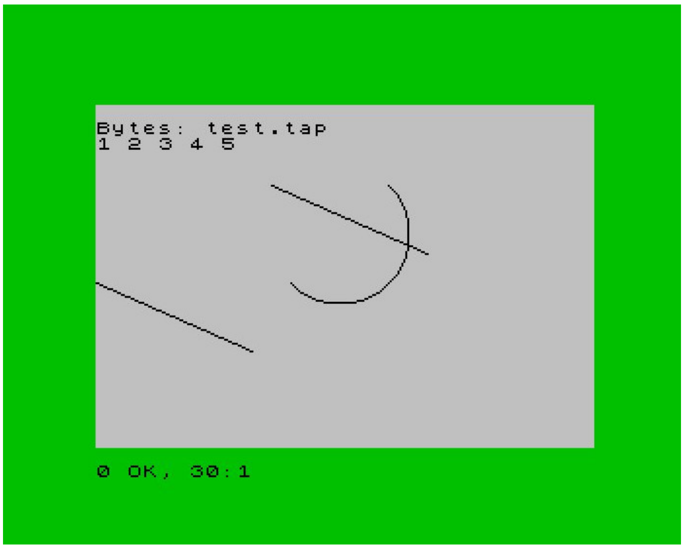
Now that I have a reasonable grasp on converting Spectrum ROM to T/S 2060 ROM, I wondered if it is possible to convert the ZX BASIC compiler to support the T/S 2068.

I've used the compiler and found that some Spectrum programs will work on the T/S 2068. As long as a ROM call is not used, it's fine. But some of the simple stuff in BASIC will not work, like CLS.

I dug through the source code files on ZX BASIC and found the library and library-asm directories. These contain files that reference the Spectrum ROM. I went through these files and converted the ROM calls to T/S 2068 equivalent calls.

```
border 4

for x = 1 to 5
   print x;" ";
next x

plot 0,100
draw 80,-35
plot 90,150
draw 80,-35

plot 100,100
draw 50,50,PI
```

I wrote some test programs that made use of these calls and they worked on the T/S 2068.



```
Bytes: test.tap
1 2 3 4 5

0 OK, 30:1
```

I have published the changed files on my web page. To get ZX BASIC to support the T/S 2068, all one needs to do is to the following:

1. Install ZX BASIC in a directory
2. Drill down in to the source code
3. Copy the files from my website over the existing files.

Now when the compiler is run, the programs will run on the T/S 2068. If you still want to compile for the Spectrum, then just install another copy of ZX BASIC in another directory and there you go.

zxpaint2

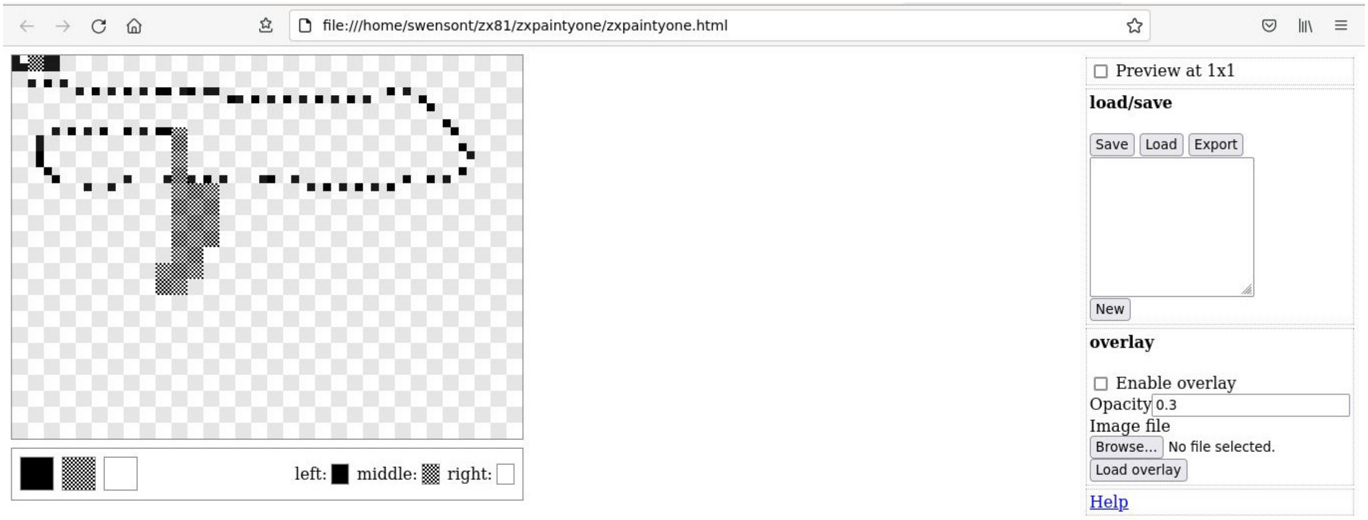
ZXpaintyOne is a browser-based tool, written in Javascript, that lets users create screen art for the ZX81. It supports the graphics characters, letting the user draw on the screen with a mouse. It is sort of like the MS Paint for the ZX81.

Once done drawing, the "save" button creates a list of the hex codes for the characters on the screen. The problem is getting those hex codes into a format that can be used on the ZX81.

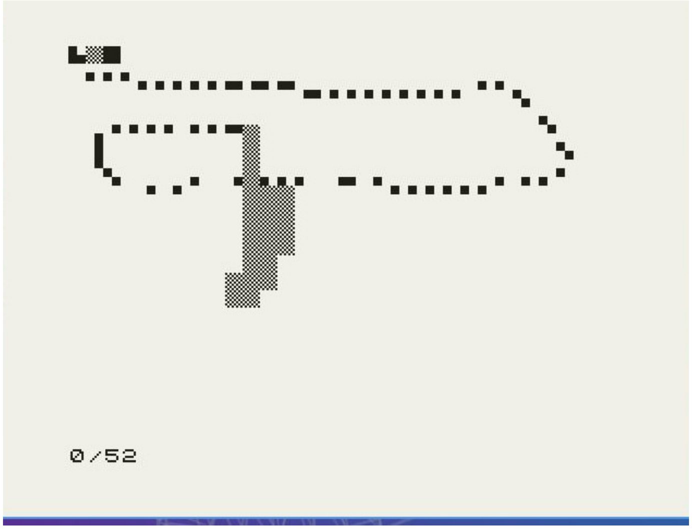
A while back I wrote zxpaint (now renamed to zxpaint1). This is a simple C program that converts the text listing of hex codes into a binary file of the same hex codes that can be loaded into screen memory of a ZX81 (using a version of load from ZXPand that can take an argument of a file).

This time I wanted to draw a screen and then use it with zxtext2p. zxtext2p has some text versions of the ZX81 graphics characters. I needed to convert the hex code into the format used by zxtext2p.

zxpaint2 is such a program. It takes two arguments, the first being the input file with the hex codes and the second is the output file in BASIC. For each line on the screen, the PRINT command is listed and then the graphics.


file:///home/swensont/zx81/zxpaintyone/zxpaintyone.html
□ Preview at 1x1
load/save
Save
Load
Export
New
overlay
□ Enable overlay
Opacity 0.3
Image file
Browse... No file selected.
Load overlay
Help
left: ■ middle: ▨ right: □

To create the input file, use the "save" option in ZXpaintyone. Then highlight the entire save section. Use CTRL-C to copy the text. Now open a new text file and use CTRL-V to past the text into the file. Do not edit the file and just save it.



zxpaint2 will create a 22 line file with BASIC statements for the screen that was created. If only part of the screen was created, then just delete those lines of BASIC that are not needed.

To add text to the screen, the lines of BASIC can be edited, or the text can be printed on the screen using the AT function. This does mean that the text will appear after the screen is drawn.

The source code is in the zip file released with this issue. It is a simple C program and in the comments is the command to compile the program. It should compile with pretty much any C compiler.

The program compiles as zxpaint2.exe. If compiling with Linux, make sure to make the program executable:

```
chmod 777 zxpaint2.ext
```

The program is run like this:

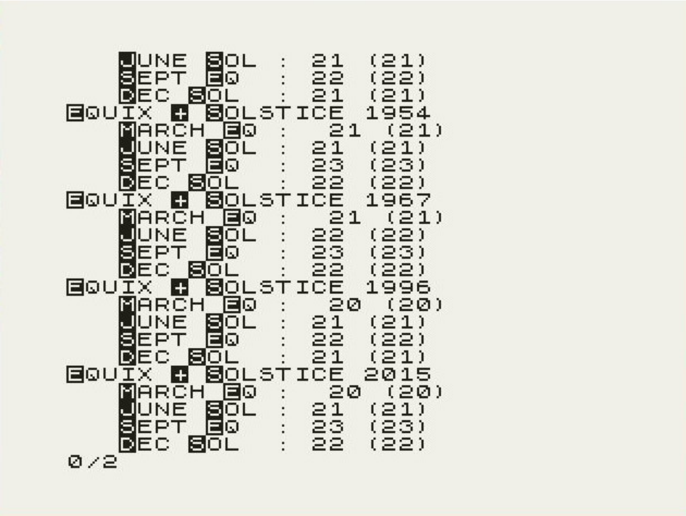
```
./zpaint2.ext file.bin file.bas
```

file.bin is a text file that has the output from ZXpaintyone saved in it. file.bas should be a new file that does not exist. Once the .bas file is created, it can be run through zxtext2p to show that it works. It can then be cut-n-pasted into the final BASIC program.

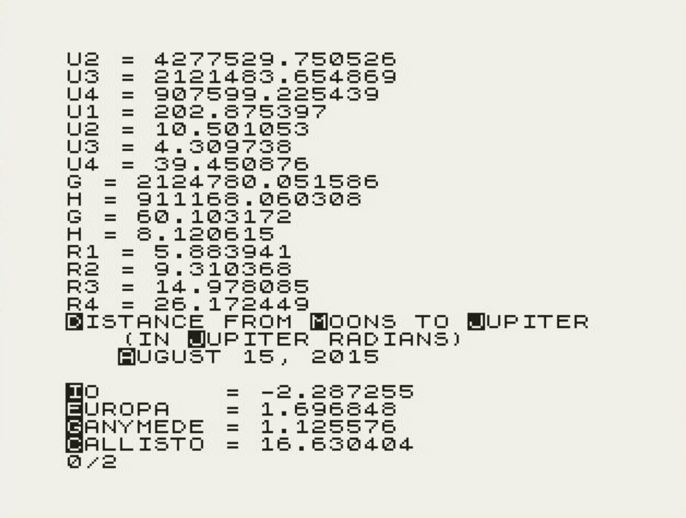
More Astronomical Programs on the ZX81

I've been reading a number of books on astronomical algorithms and have been writing programs in C for each of the algorithms. I originally worked on the QL, but I found it easier to work on the code in Linux and then port to the QL. Over time I have a number of programs. An example is that I have found a number of ways to determine the date of Easter and written programs for each one. The programs are written as a function or a procedure and then I have a test program that calls the function or procedure.

Since it is all in C, I wanted to see how easy it would be to just compile the programs with Z88DK. How many changes from the original code would be necessary. For most programs, there were no changes to be made. The code that compiled on Linux with GCC also compiled with Z88DK. I did run in to a few programs that failed after they were compiled. After some head scratching and then going to my backup C expert, I found the issue to be one of type sizes. The INT size between Linux and Z88DK was different, so I was getting overflows on the ZX81. After a few code changes, the programs were compiling and running just fine.



```
   JUNE SOL : 21 (21)
   SEPT EQ  : 22 (22)
   DEC SOL  : 21 (21)
EQUIX + SOLSTICE 1954
   MARCH EQ :  21 (21)
   JUNE SOL : 21 (21)
   SEPT EQ  : 23 (23)
   DEC SOL  : 22 (22)
EQUIX + SOLSTICE 1967
   MARCH EQ :  21 (21)
   JUNE SOL : 22 (22)
   SEPT EQ  : 23 (23)
   DEC SOL  : 22 (22)
EQUIX + SOLSTICE 1996
   MARCH EQ :  20 (20)
   JUNE SOL : 21 (21)
   SEPT EQ  : 22 (22)
   DEC SOL  : 21 (21)
EQUIX + SOLSTICE 2015
   MARCH EQ :  20 (20)
   JUNE SOL : 21 (21)
   SEPT EQ  : 23 (23)
   DEC SOL  : 22 (22)
0/2
```



```
U2 = 4277529.750526
U3 = 2121483.654869
U4 = 907599.225439
U1 = 202.875397
U2 = 10.501053
U3 = 4.309738
U4 = 39.450876
G = 2124780.051586
H = 911168.060308
G = 60.103172
H = 8.120615
R1 = 5.883941
R2 = 9.310368
R3 = 14.978085
R4 = 26.172449
DISTANCE FROM MOONS TO JUPITER
   (IN JUPITER RADIANS)
   AUGUST 15, 2015

IO        = -2.287255
EUROPA    = 1.696848
GANYMEDE  = 1.125576
CALLISTO  = 16.630404
0/2
```

It was nice to see that simple C programs written on Linux would easy work on the ZX81. To port to the QL, the only changes were putting in the locations of where the source was located (win2_ vs win1_).

I have zipped up the programs (source and executable) and added them to the zip file associated with this issue.

Prime Number Benchmark

In the January 1990 issue of the Capital Area Timex Sinclair (CATS) Users Group newsletter, Duane Parker published a prime number program that will generate prime numbers from 32767 to X. The program was used as a way to test the speed differences between languages and compilers on the QL. Duane provided examples in Forth, Pascal and SuperBasic. A few months later Herb Schaaf provided an example in C with the Small-C compiler. A while after that I ported the program to BBC Basic for the Z88.

Recently I was digging thought some CATS newsletters and ran across the benchmark and thought that I would try it on the ZX81 and T/S 2068. I ported the BASIC version and tried it out on some different emulators. I took the C version, took out some uneeded bits and ported it to Z88DK. I was then able to compile for the ZX81 and T/S 2068. For the T/S 2068, I also did a version for Boriel's ZX Basic compiler.

The original benchmark was from from 32767 to 29000. On a standard QL, that run took 245 seconds. I tried the same run on the ZX81 using the sz81 emulator and gave up after 30 minutes. I decided to adjust the number and went from 29000 to 32000. Here is the times that I saw:

ZX81

BASIC
  sz81   - 10 min 35 sec
  Zesarux - 9 min 22 sec
  ts1000 - 4 min 3 sec
  Xtender - 5 min 0 sec
  zx81em - 18 sec

Z88DK (C)
  sz81   - 23 sec
  Zesurux - 20 sec
  ts1000 - 9 sec
  Xtender - 11 sec
  zx81em - n/a

It is interesting to see how much faster using C is when compared to BASIC.

On all of the emulators, I did not adjust the speed of the emulator. Both sz81 and Zesarux were at 100% speed and I could have set them to 4x or more.

The three other emulators were DOS emulators that I ran in DOSBOX. I also did not adjust the speed of DOSBOX. The ts1000 emulator is by Jeff Vavasour. Xtender is by Carlos Delhez. zx81em is by Paul Robson and is a very fast emulator.

For the T/S 2068, I only had Zesarux to use. Here are the times that I found:

BASIC
  Zesarux - 2 min 52 sec
Z88DK
  Zesarux - 6 sec
ZX BASIC
  Zesarux - 16 sec

The ZX Basic compiler is not as good as using C, but it is far faster than using just BASIC.

This benchmark is a good way to see how much faster a program can be when it is compiled and that C is faster than compiled BASIC.

Structured Programming with ZX81 BASIC

The ZX81 was my first computer. I learned a fair bit about programming using the ZX81. ZX81 BASIC is limited in its scope, even more limited than BASIC on the Spectrum. ZX81 BASIC was designed to be less complex and a starting platform for programmers.

In college, Pascal was the primary language and I leared about structured programming using Pascal. When I bought the QL, I was able to use what I learned with Pascal with SuperBasic. SuperBasic had all the constructs that I had learned with structured programming.

As I think about writing software for the ZX81, the difficulty I have is that I think in structured programming and ZX81 does not support structured programming. An IF statement only allows one command and not a block of commands. There is no support for procedures. I find it really limiting.

In doing some research when started working on this article, I found that there is a precedence in teaching structured programming with BASIC. A number of books have been written on the subject, back when BASIC was the primary language for microcomputers:

"Structured BASIC Programming", Kemery & Kurtz, 1987
"Introduction to Structured Programming using BASIC", Barnett, 1984

In Creative Computing, May - Sept. 1984, Arthur Luehrmann wrote a series of articles on structured programming with BASIC.

This article is my way of reminding myself how to implement the different control and iteration structures in ZX81 Basic. I can first write the code in a structured way and then edit to fit ZX81 Basic.

Control Structures

IF ... THEN

The IF .. THEN structure is allowed on the ZX81, but it only allows a single command after THEN. Most languages allow a block of commands after the THEN.

```
IF <condition> THEN
    ........
    ........
    ........
END IF
```

```
100 IF <condition> THEN GOTO 120
110 GOTO 150
120  .........
130  .........
140  .........
150  <continue with program>
```

IF ... THEN ... ELSE

ZX81 Basic does not support the ELSE keyword, but an implied ELSE can be used.

```
IF <condition> THEN
  .........
  .........  <block 1>
  .........
ELSE
  .........
  .........  <block 2>
  .........
END IF
```

```
100 IF <condition> THEN GOTO 160
110 REMARK ELSE
120 .........
130 .........  <block 2>
140 .........
150 GOTO 200
160 .........
170 .........   <block 1>
180 .........
200 <continuation of program>
```

CASE/SWITCH/SELECT

Known as CASE in Pascal, SWITCH in C, and SELECT in SuperBasic, this structure allows branching based on different values of a single variable.

```
SELECT ON var
  = 1
    ........
    ........
  = 2
    ........
    ........
  = 3
   ........
   ........
  = REMAINDER
    ........
    ........
END SELECT
```

```
100 IF CHOICE = 1 THEN GOTO 160
110 IF CHOICE = 2 THEN GOTO 190
120 IF CHOICE = 3 THEN GOTO 220
130 REMARK REMAINDER
140 .........
150 .........
160 GOTO 250
170 .........
180 .........
190 GOTO 250
200 .........
210 .........
220 GOTO 250
230 .........
240 .........
250 <continuation of program>
```

Iteration Structures

FOR ... NEXT

The ZX81 supports the FOR..NEXT loop structure. No change is needed for this.

WHILE

The WHILE structure is for controlling loops not based on a number (like the FOR..NEXT) but based on a condition.

```
WHILE ( X < 200)
   ........
   X = X * 2
   ........
END WHILE
```

```
100 IF X >= 200 then goto 500
110   .......
120   X = X * 2
130   .......
140   GOTO 100
500   < continuation of program >
```

In some languages this structure has two type:

```
WHILE (condition) DO

DO...
  .......
WHILE (condition)
```

In the first structure the check if made at the start of the code and in the second case the check is make at the bottom of the code. The main difference is that in the second case, the program will pass through the code at least once, where in the first case, it will not pass if the condition is not met.

Procedures

A procedure is a piece of code that can be called from many places in the main code. Written once, it can be used many times from many places.

```
DEFINE PROCEDURE foo  ( var )
   ..........
   ..........
   ..........
END DEFINE foo
```

ZX81 Basic has GOSUB, short for Go Subroutine, which is similar to a procedure. A procedure usually has variables. In ZX81 Basic certain variables need to be set first before calling the subroutine. These need to be documented.

```
90  LET var = 15
100 GOSUB 500
 .....
 .....
500 .........
510 .........
520 .........
530 RETURN
```

Functions

A function would be the same on the ZX81 as a procedure. All variables will be global and the return variable (implied) will have to be documented so it is known what variable will hold the final result.

```
Define Function Foo (var)
  .......
  .......
  return x
end define foo


100 let var = 53
110 let x = 0
120 gosub 500
130 .......

500 let var = var + 1
510 let x = var
520 return
```